



**HAL**  
open science

# Incremental algorithms for computing the set of period sets

Eric Rivals

► **To cite this version:**

| Eric Rivals. Incremental algorithms for computing the set of period sets. 2024. lirmm-04531880v1

**HAL Id: lirmm-04531880**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-04531880v1>**

Preprint submitted on 4 Apr 2024 (v1), last revised 27 Sep 2024 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

# 1 Incremental algorithms for computing the set of 2 period sets

3 Eric Rivals   

4 LIRMM, Université Montpellier, CNRS, Montpellier, France

## 5 — Abstract —

---

6 Overlaps between strings are crucial in many areas of computer science, such as bioinformatics, code  
7 design, and stringology. A self overlapping string is characterized by its periods and borders. A  
8 period of a string  $u$  is the starting position of a suffix of  $u$  that is also a prefix  $u$ , and such a suffix is  
9 called a border. Each word of length, say  $n > 0$ , has a set of periods, but not all combinations of  
10 integers are sets of periods. The question we address is how to compute the set, denoted  $\Gamma_n$ , of all  
11 period sets of strings of length  $n$ . Computing the period set for all possible words of length  $n$  is  
12 clearly prohibitive. The cardinality of  $\Gamma_n$  is exponential in  $n$ . One dynamic programming algorithm  
13 exists for enumerating  $\Gamma_n$ , but it suffers from an expensive space complexity. After stating some  
14 combinatorial properties of period sets, we present a novel algorithm that computes  $\Gamma_n$  from  $\Gamma_{n-1}$ ,  
15 for any length  $n > 1$ . The period set of a string  $u$  is a key information for computing the absence  
16 probability of  $u$  in random texts. Hence, computing  $\Gamma_n$  is useful for assessing the significance of  
17 word statistics, such as the number of missing  $k$ -mers in a random text, or the number of shared  
18  $k$ -mers between two random texts. Besides applications, investigating  $\Gamma_n$  is interesting per se as it  
19 unveils combinatorial properties of string overlaps.

20 **2012 ACM Subject Classification** Mathematics of computing → Discrete mathematics

21 **Keywords and phrases** autocorrelation, string, period, combinatorics, periodicity

22 **Digital Object Identifier** 10.4230/LIPIcs.???.2024.

23 **Funding** E. Rivals is supported by the European Union's Horizon 2020 research and innovation  
24 programme under the Marie Skłodowska-Curie grant agreement No 956229.



© Eric Rivals;  
licensed under Creative Commons License CC-BY 4.0

???

Editors: ???; Article No. ; pp. :1–22



Leibniz International Proceedings in Informatics  
LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

25 **1** Introduction

26 Considering finite words or strings over a finite alphabet, we say that a word  $u$  overlaps a  
 27 word  $v$  if a suffix of  $u$  of length, say  $i$ , equals a prefix of  $v$  of the same length. A pair of words  
 28  $u, v$  can have several overlaps of different lengths. For instance, over the binary alphabet  
 29  $\{a, b\}$ , consider the words  $u := ababba$  and  $v := abbabb$ :  $u$  overlaps  $v$  with the suffix-prefix  
 30  $abba$ , and with  $a$ . It appears that the longest overlap contains all other overlaps: to find  
 31 all overlaps from  $u$  to  $v$ , it suffices to study the overlaps of  $abba$  with itself. This is true in  
 32 general for any pair of words.

33 For a word  $u$ , a suffix that equals a prefix of  $u$  is called a border, and the length of  $u$   
 34 minus the length of a border, is called a period. Computing all self-overlaps of a word  $u$  is  
 35 computing all its borders or all its periods, which can be done in linear time (see [28]). For  
 36 instance the word *abracadabra* of length  $n = 11$  has the following set of periods  $\{0, 7, 10\}$   
 37 (zero being the trivial period - the whole word matches itself). This problem and variants of  
 38 it have been widely studied, since it is useful in the design of pattern matching algorithms  
 39 (like [12]).

40 The reader can easily convince her/himself that distinct words of the same length can  
 41 share the same set of periods, even if one forbids a permutation of the alphabet. For a word  $u$ ,  
 42 let us denote by  $P(u)$  its period set (which we abbreviate by PS). In this work, we investigate  
 43 algorithms to enumerate all possible period sets for any words of a given length  $n$ . This set is  
 44 denoted  $\Gamma_n$  for  $n > 0$  and is non trivial if the alphabet contains at least two symbols. Brute  
 45 force enumeration can consider all possible words of length  $n$  and compute their period set,  
 46 but this approach obviously becomes computationally unaffordable for  $n > 30$ .

47 Interest in  $\Gamma_n$  sparkled mostly in the 80's, when researchers started to evaluate the  
 48 average behavior of pattern matching algorithms, or that of filtering strategies for sequence  
 49 alignment, text comparisons or clustering. A powerful filtering when comparing two texts, is  
 50 to list their  $k$ -mers, for appropriate values of  $k$ , and then compute e.g. a Jaccard distance  
 51 between their  $k$ -mer spectrum, to see whether the two texts are similar enough to warrant a  
 52 costly alignment procedure [31].

53 In a different area, testing Pseudo-Random Number Generators can also be translated  
 54 into a question on vocabulary statistics. Indeed in truly random real numbers written as  
 55 sequence of digits, all substrings of a given length, say  $k$ , should ideally have an almost equal  
 56 number of occurrences. In other words, for any substring the number of its occurrences in  
 57 the sequence should not significantly deviate from a theoretical expectation. Empirical tests,  
 58 named *Monkey Tests*, were developed for such generators [17, 20, 14]. It turns out that the  
 59 absence probability of a word/string in a random text is essentially controlled by the period  
 60 set of the word [8]. Hence, the need for enumerating  $\Gamma_n$  appears in diverse domains of the  
 61 literature [21, 22].

62 The question of enumerating  $\Gamma_n$  is non trivial since  $\Gamma_n$  grows exponentially, as shown  
 63 in [8], which provided the first upper and lower bound on the logarithm of its cardinality,  
 64 which is denoted  $\kappa_n$ . The sequence of integers formed by  $\kappa_n$  in function of string length  
 65  $n$  has an entry in the OEIS. Even the most recent asymptotic upper and lower bounds of  
 66  $\log(\kappa_n)/\log_2(n)$  are not close to known values of this ratio. At least, the convergence of  
 67 this ratio, which was conjectured in 1981, was recently proven in 2023 [25]. Currently, only  
 68 a dynamic programming algorithm exists to enumerate  $\Gamma_n$ , but it suffers from high space  
 69 complexity [24].

70 In this work, we propose an incremental approach that computes  $\Gamma_n$  from  $\Gamma_{n-1}$  and uses  
 71 linear space in  $n$ . Our approach needs a *certification* function, which can tell if a subset of

$\{0, 1, \dots, n-1\}$  is a valid period set or not. Three incremental algorithms that differ by their certification function are presented. They allow one to compute  $\Gamma_n$  for some values considered in real world applications, and thus to investigate combinatorial and statistical properties of  $\Gamma_n$  and of its cardinality.

**Plan.** In Section 2, we introduce a notation, preliminary results, and review some known results. In section 3, we present the general framework of the incremental algorithm for computing  $\Gamma_n$ , and two variants of it. In section 4 an algorithm for binary realization of period set is explained; it can also be used in the incremental algorithm. In section 5, notions of fate of a period set are defined. Finally, in section 6, we show visualization of  $\Gamma_n$  as a lattice to illustrate these notions and plots interesting parameters related to  $\Gamma_n$ , before concluding with open questions.

## 2 Related works, notation and preliminary results.

### 2.1 Notation

Here we introduce a notation and basic definitions.

For two integers  $p, q \in \mathbb{N}_{>0}$ , we denote the fact that  $p$  divides  $q$  by  $p \mid q$  and the opposite by  $p \nmid q$ . We consider that strings and arrays are indexed from 0. We use  $=$  to denote equality, and  $:=$  to denote a definition.

An *alphabet*  $\Sigma$  is a finite set of *letters*. A finite sequence of elements of  $\Sigma$  is called a *word* or a *string*. The set of all words over  $\Sigma$  is denoted by  $\Sigma^*$ , and  $\varepsilon$  denotes the empty word (the only word on length 0). For a word  $x$ ,  $|x|$  denotes the *length* of  $x$ . Let  $n$  be an integer. The set of all words of length  $n$  is denoted by  $\Sigma^n$ . Given two words  $x$  and  $y$ , we denote by  $x.y$  the *concatenation* of  $x$  and  $y$ . For a word  $w$ , we define the powers of  $w$  inductively:  $w^0 := \varepsilon$  and for any  $n > 0$ ,  $w^n := w.w^{n-1}$ . A word  $u$  is *primitive* if there exists no word  $v$  such that  $u = v^k$  with  $k \geq 2$ .

Let  $u := u[0..n-1] \in \Sigma^n$ . For any  $0 \leq i \leq j \leq n-1$ , we denote by  $u[i-1]$  the  $i^{\text{th}}$  symbol in  $u$ , and by  $u[i..j]$ , the substring starting at position  $i$  and ending at position  $j$ . In particular,  $u[0..j]$  denotes a prefix and  $u[i..n-1]$  a suffix of  $u$ .

Let  $x, y \in \Sigma^*$  and let  $j$  be an integer such that  $0 \leq j \leq |x|$  and  $j \leq |y|$ . If  $x[n-j..|x|-1] = y[0..j-1]$ , then the *merge* of  $x$  and  $y$  with offset  $j$ , which is denoted by  $x \oplus_j y$ , is defined as the concatenation of  $x[0..n-j-1]$  with  $y$ . I.e.,  $x \oplus_j y := x[0..n-j-1]y$ .

#### 2.1.1 Periodicity

In this subsection, we define the concepts of period, period set, basic period, and autocorrelation, and then review some useful results.

► **Definition 1** (Period/border). *The string  $u = u[0..n-1]$  has period  $p \in \{0, 1, \dots, n-1\}$  if and only if  $u[0..n-p-1] = u[p..n-1]$ , i.e. for all  $0 \leq i \leq n-p-1$ , we have  $u[i] = u[i+p]$ . Moreover, we consider that  $p = 0$  is a period of any string of length  $n$ . The substring  $u[p..n-1]$  is called a *border*.*

Zero is also called the trivial period. The *period set* of a string  $u$  is the set of all its periods and is denoted by  $P(u)$ . The *weight* of a period set is its cardinality. The smallest non-zero period of  $u$  is called its *basic period*. When  $P(u) = \{0\}$ , we consider that its basic period is the string length  $n$ .

Let  $\Gamma_n := \{Q \subset \{0, 1, \dots, n-1\} \mid \exists u \in \Sigma^n : Q = P(u)\}$  be the set of all period sets of strings of length  $n$ . We denote its cardinality by  $\kappa_n$ . The period sets in  $\Gamma_n$  can be partitioned

## XX:4 Incremental algorithms for the set of period sets

115 according to their basic period; thus, for  $0 \leq p < n$ , we denote by  $\Gamma_{n,p}$  the subset of period  
116 sets whose basic period is  $p$ , and by  $\kappa_{n,p}$  the cardinality of this subset.

117 For the most important properties on periods, we refer the reader to [8, 15] and Ap-  
118 pendix F.1. Below we recall the characterization of period sets from [8], and state the version  
119 for strings of the famous Fine and Wilf (FW) theorem [6]. We refer the reader to [24, 25] for  
120 more properties on period sets.

### 121 2.2 Related works

122 The notion of overlap in strings is crucial in many areas and applications, among others:  
123 combinatorics, bioinformatics, code design, or string algorithms.

124 In cryptography and network communication, the so-called prefix-free, bifix-free, and  
125 cross-bifix-free codes are used for synchronization purposes. Their design require to select  
126 a set of words that are mutually non overlapping (aka unbordered). This topic has been  
127 studied for long: the seminal construction algorithm from Nielsen was published in 1973 [18]  
128 together with a note on the expected duration of a search for a fixed pattern in random data  
129 [19], thereby linking explicitly pattern matching and code design. Improved algorithms for  
130 such code design were published until recently e.g. [2, 1].

131 Many aspects of periodicity of words were and are still extensively studied in combinatorics  
132 giving rise to a huge literature [5, 7]. For instance, the periods of random words were  
133 investigated in [11] and the Fine and Wilf (FW) theorem was generalized to more than two  
134 words [4]. Some literature has been devoted to constructing extremal FW-words relative to  
135 a subset of periods: Given a set of integers  $R$ , the question is find the longest word  $w$  such  
136 that the period set of  $w$  includes  $R$ , but does not include the gcd of periods in  $R$ . Algorithms  
137 were proposed and gradually improved in [29, 30] among others. This question is related to  
138 our question regarding the fate of period set when the string length  $n$  increases.

139 In bioinformatics, string overlaps are central in the question of DNA or genome assembly.  
140 When a genome is broken into pieces and then sequenced, one gets hundreds of millions  
141 of reads, which are strings over a 4-letter alphabet. One then needs to compute overlaps  
142 between reads, represent these overlaps in graph and search for a Hamiltonian or Eulerian  
143 path (depending on the graph) satisfying some length or overlap conditions to infer the full  
144 sequence of the genome; see [9, 16] for more pointers. The comparison of sequences and the  
145 statistical issues regarding inference of motifs, which both look at the set of  $k$ -mers occurring  
146 in sequences, are also related to periodicity [27, 26, 31].

147 Now let us review the most closely related literature to our question. In a seminal work  
148 [8], Guibas and Odlyzko defined the notion of autocorrelation of  $u$ , which encodes the period  
149 set in a binary vector of length  $n$ , where 1 at position  $i$  indicates that  $i$  is a period of  $u$ .  
150 The binary encoding gives the length  $n$ , but is otherwise equivalent to the notion of set  
151 period <sup>1</sup>. They have exhibited a recursive characterization of an autocorrelation, which  
152 runs in linear time in  $n$ . They have provided lower and upper bounds for  $\log(\kappa_n)/\log_2(n)$ ,  
153 and conjectured that their lower bound was also an upper bound. They also proposed an  
154 algorithm to compute the number of strings in  $\Sigma^n$  that share the same period set, which they  
155 termed the *population* of a period set. A key result of their work is the *alphabet independence*  
156 of  $\Gamma_n$ : Any alphabet of size at least two gives rise to the same set of period sets, i.e., to  $\Gamma_n$ .  
157 Of course, if the alphabet is a singleton all questions mentioned here become trivial. In the

---

<sup>1</sup> We will see in 5 that a period set can belong to several  $\Gamma_n$  for several values of  $n$ .

158 sequel, let us consider that  $\text{card}(\Sigma) > 1$ . Note that their characterization of period sets was  
 159 re-discovered a few years later [13].

160 Circa 20 years later, Halava et al. gave a simpler proof of the alphabet independence  
 161 result [10] by solving the following question. For any period set  $Q$  of  $\Gamma_n$ , let  $v$  be a word over  
 162 an alphabet of cardinality larger or equal to 2 such that  $P(v) = Q$ ; then compute a binary  
 163 word  $u$  such that  $P(u) = Q$ . Indeed, they exhibited a linear time algorithm that computes  
 164 such a word  $u$  from  $v$ .

165 At the same period, other authors investigated the structure of  $\Gamma_n$  to show that it is a  
 166 lattice that does not satisfy the Jordan-Dedekind condition [24]. Moreover, they designed  
 167 an enumeration algorithm for  $\Gamma_n$  that uses a dynamic programming approach. Given that  
 168  $\kappa_n$  is exponential in  $n$ , their enumeration algorithm also is, but its main drawback lies in  
 169 memory usage, which requires to store all  $\Gamma(i)$  for  $0 < i \leq \lfloor 2n/3 \rfloor$ . With combinatorial  
 170 arguments about the number of binary partitions of an integer, they provided improved  
 171 the lower bounds for  $\log(\kappa_n)/\log_2(n)$ . Many concepts and results have been extended  
 172 to words with don't care symbols, e.g. [3]. In 2023, the conjecture stated by Guibas and  
 173 Odlyzko regarding this ratio was finally proven to be correct, thereby implying that for  
 174  $\log(\kappa_n)/\log_2(n)$  converges towards  $1/(2\log(2))$  when  $n$  tends to infinity [25].

### 175 2.3 Rule based characterization

176 In their seminal paper, Guibas and Odlyzko characterized autocorrelations by three conditions:  
 177 they must

- 178 1. start with a 1 (i.e., zero is a period)
- 179 2. satisfy the Forward Propagation Rules (FPR)
- 180 3. satisfy the Backward Propagation Rule (BPR)

181 Let us formulate the FPR and BPR in terms of sets (rather than in term of binary vector  
 182 as in [8]). Let  $P$  a subset of  $\{0, 1, \dots, n-1\}$ .

183 ► **Definition 2.**  $P$  satisfies the FPR iff for all pairs  $p, q$  in  $P$  satisfying  $0 \leq p < q < n$ , it  
 184 follows that  $p + i(q-p) \in P$  for all  $i = 2, \dots, \lfloor (n-p)/(q-p) \rfloor$ .

185 ► **Definition 3.**  $P$  satisfies the BPR iff for all pairs  $p, q$  in  $P$  satisfying  $0 \leq p < q < 2p$ , and  
 186  $(2p-q) \notin P$ , it follows that  $p-i(q-p) \notin P$  for all  $i = 2, \dots, \min(\lfloor p/(q-p) \rfloor, \lfloor (n-p)/(q-p) \rfloor)$ .

187 In [8], the authors give a version for strings of the famous Fine and Wilf theorem [6],  
 188 a.k.a. the periodicity lemma. A nice proof was provided by Halava and colleagues [10].

189 ► **Theorem 4 (Fine and Wilf).** Let  $p, q$  be periods of  $u \in \Sigma^n$ . If  $n \geq p + q - \text{gcd}(p, q)$ , then  
 190  $\text{gcd}(p, q)$  is a period of  $u$ .

191 We can reformulate this theorem as a condition that must be satisfied by a period set  $P$  of  
 192  $\Gamma_n$ .

193 ► **Theorem 5.** Let any pair  $p, q$  of periods of  $P$  such that  $\text{gcd}(p, q) \notin P$ , and define  
 194  $FW(p, q) := p + q - \text{gcd}(p, q)$ . Then  $FW(p, q)$  must be strictly larger than  $n$ .

195 We call  $FW(p, q)$  the *Fine and Wilf (FW) limit* of  $(p, q)$ , and the fact that  $FW(p, q)$  must  
 196 be larger than  $n$ , the *FW condition*.

197 We define the notion of *nested set*. It helps formulating definitions and properties that  
 198 were originally expressed as suffixes of autocorrelations in [8].

## XX:6 Incremental algorithms for the set of period sets

199 ► **Definition 6.** Let  $n > 0$  and  $P$  be a subset of  $\{0, 1, \dots, n-1\}$ . Let  $q$  be an element of  $P$ .  
 200 Let us denote the nested set of  $P$  starting at period  $q$  as  $P_q$ :

$$201 \quad P_q := \{(r - q) \text{ for each } r \in P \text{ such that } r \geq q\}.$$

202 By construction  $P_q$  starts with 0; moreover, if we choose  $q = 0$  then  $P_q = P$ . Now assume  
 203 that  $P$  is valid PS of length  $n$ , and  $q$  a period of  $P$ . Then, by Theorem 9 (which we recall in  
 204 Section 3), we have that  $P_q$  is a valid PS of length  $(n - q)$ .

### 205 2.4 Checking the FPR and the BPR

206 Let  $n > 0$  and  $P$  be a subset of  $\{0, 1, \dots, n-1\}$ . We assume that  $P$  is given as an ordered  
 207 array. The complexity for checking the FPR or the BPR for  $P$ , has, to our knowledge not  
 208 been previously addressed. For any pair  $p < q$ , we call their difference  $(q - p)$ , an offset.

209 **Checking the BPR.** Here, we demonstrate a property that relates the BPR to the  
 210 FW theorem. Precisely, if BPR is violated for some pair  $(p, q)$  at length  $n$ , with period  
 211  $r := p - i(q - p)$  for some  $i$ , then the pair  $(p - r, q - r)$  violates the FW condition of Theorem 5  
 212 in the nested set of length  $(n - r)$ .

213 ► **Lemma 7.** Let  $(p, q)$  be a pair of integers that violate the BPR, and let  $i \geq 2$  such that  
 214  $r := p - i(q - p) \in P$ . Then the pair  $(p - r, q - r)$  violates the FW condition for length  $(n - r)$ .

215 **Proof.** Let  $P \in \Gamma_n$ . Let  $p, q$  in  $P$  satisfying  $0 \leq p < q < 2p$  be such that  $(2p - q) \notin P$ . Assume  
 216  $(p, q)$  violates the BPR. Then, there exists  $i$  in  $[2, \dots, \min(\lfloor p/(q - p) \rfloor, \lfloor (n - p)/(q - p) \rfloor)]$ ,  
 217 such that  $p - i(q - p) \notin P$ . If several such integers exist, choose  $i$  as their minimum, and  
 218 define  $r := p - i(q - p)$ . We will show that the nested period set of  $P$  for length  $(n - r)$   
 219 is not valid since two of its periods violates the FW condition, which would require their  
 220 gcd as an additional period, thereby implying that  $P$  is not a valid period set for length  
 221  $n$ , a contradiction. Since  $i$  is chosen minimal, we have that  $\gcd(p, q)$  is not in  $P$  by the  
 222 definition of the BPR. Note that  $p - r = i(q - p)$  and  $q - r = (i + 1)(q - p)$ . Thus, one  
 223 gets  $\gcd(p - r, q - r) = q - p$ , and the FW limit of  $(p - r, q - r)$  equals  $2i(q - p)$ . Indeed,  
 224  $FW(p - r, q - r) := p - r + q - r - \gcd(p - r, q - r) = 2p - 2r = 2i(q - p)$ . By hypothesis,  
 225 we have:

$$\begin{aligned} & i && \leq && (n - p)/(q - p) \\ \Leftrightarrow & p + i(q - p) && \leq && n \\ 226 \quad \Leftrightarrow & r + 2i(q - p) && \leq && n \\ & \Leftrightarrow && FW(p - r, q - r) && \leq && n - r \end{aligned}$$

227 meaning that  $(p - r, q - r)$  violates the FW condition of Theorem 5 for length  $(n - r)$ . ◀

228 In algorithmic terms, checking the BPR can be done by checking the FW condition of  
 229 Theorem 5 in each nested set.

230 **Checking the FPR.** Some properties are explained in Appendix D and lead to this  
 231 Lemma.

232 ► **Lemma 8.** Let  $P$  is a subset of  $[0, \dots, n-1]$ , that is ordered, and has zero as first period.  
 233 Checking the FPR for  $P$  in general takes at most  $O(n \log_2(n))$  time.



### 3 Incremental enumeration framework

#### 3.1 Rationale for an incremental algorithm to enumerate $\Gamma_n$

The rule based characterization of autocorrelations from [8] implies a *special substring* property. Indeed, Lemma 3.1 in [8] states: If a binary vector  $v$  satisfies the forward and backward propagation rules, then so does any prefix or suffix of  $v$ . As the characterization also requires that an autocorrelation has its first bit equal to one, or equivalently that zero belongs to any period set, one gets the following theorem.

► **Theorem 9.** *Let  $v$  be an autocorrelation of length  $n$ . Any substring  $v_i \dots v_j$  of  $v$  with  $0 \leq i \leq j < n$  such that  $v_i = 1$  is an autocorrelation of length  $j - i + 1$ .*

Applying Theorem 9 to a prefix of  $v$ , one gets for any  $n > 0$ : The prefix of length  $(n - 1)$  from an autocorrelation of length  $n$  is an autocorrelation of length  $(n - 1)$ . In terms of period sets, this statement can be reformulated as:

► **Corollary 10.** *If  $P$  is a period set of  $\Gamma_n$ , then  $P \setminus \{n - 1\}$  belongs to  $\Gamma_{n-1}$ .*

First, this means that, knowing  $\Gamma_n$ , it is easy to compute  $\Gamma_{n-1}$ . It suffices to consider each element of  $\Gamma_n$  in turn (or in parallel) and to eventually remove the period  $(n - 1)$  from it (i.e., if  $(n - 1)$  belongs to it) to obtain an element of  $\Gamma_{n-1}$ . With this procedure one can obtain the same element of  $\Gamma_{n-1}$  twice, and one must keep track of this to avoid redundancy. Conversely, we get the Lemma that underlies the incremental approach for computing  $\Gamma_n$ :

► **Lemma 11.** *Let  $P$  be a period set of  $\Gamma_{n-1}$ . Then, a period set  $Q$  of  $\Gamma_n$  can only be of two alternative forms: either  $P$  or  $P \cup \{n - 1\}$ .*

#### 3.2 Incremental algorithm framework

Lemma 11 suggests an approach for computing  $\Gamma_n$  using  $\Gamma_{n-1}$ . Consider each  $P$  from  $\Gamma_{n-1}$ , and certify that the candidate sets,  $P$  and  $P \cup \{n - 1\}$ , are valid period sets of  $\Gamma_n$ . Clearly, Algorithm 1 presents the general incremental algorithm for  $\Gamma_n$ , where `certify` denotes the certification function used. This function must take as input  $n$  and a subset of  $\{0, 1, \dots, n - 1\}$ , and check the validity of this subset as a period set for length  $n$ .

##### Algorithm 1 IncrementalGamma

---

**Input:**  $n > 1$ : integer;  $\Gamma_{n-1}$ : the set of period sets for length  $n - 1$

**Output:**  $\Gamma_n$ : the set of period sets for length  $n$ ;

```

1  $G := \emptyset;$  //  $G$ : variable to store  $\Gamma_n$ 
2 for all  $P \in \Gamma_{n-1}$  do
3   if certify( $P, n$ ) then // check that  $P$  is valid at length  $n$ 
4     insert  $P$  in  $G$ ;
5    $Q := P \cup \{n - 1\}$  // build extension  $P$  with period  $n - 1$ ;
6   if certify( $Q, n$ ) then // check that  $Q$  is valid at length  $n$ 
7     insert  $Q$  in  $G$ ;
8 return  $G$ ;
```

---

The recursive predicate  $\Xi$  from [8] (cf. Appendix F.2) is one possible certification function, which does exactly what is required for any subset of  $\{0, 1, \dots, n - 1\}$ , in linear time [8]. This means that Algorithm 1 correctly computes  $\Gamma_n$  from  $\Gamma_{n-1}$ . However, since we know that  $P$  belongs to  $\Gamma_{n-1}$ , the candidate sets are not **any subset** of  $\{0, 1, \dots, n - 1\}$ , but specific



## XX:8 Incremental algorithms for the set of period sets

265 ones that already satisfy some constraints for length  $(n - 1)$ . Therefore, finding alternative  
266 certification functions is interesting.

267 Besides its simplicity, the main advantage of Algorithm 1, compared to the dynamic  
268 programming enumeration algorithm of [23], is its space complexity. Here, the computation  
269 considers each period set  $P$  from  $\Gamma_{n-1}$  in turn (and independently from the others), executes  
270 twice the certification function for  $P$  and  $Q$ ; this implies that the memory required, besides  
271 storage of  $P, Q$ , is the one used by the certification function. With the predicate  $\Xi$ , it is  
272 linear in  $n$ , so  $O(n)$  space. The time complexity is proportional to  $\kappa_n$  (i.e., the cardinality of  
273  $\Gamma_n$ ) times the running time of the certification function, which yields the following theorem.

274 ► **Theorem 12.** *One has*

- 275 1. *Algorithm 1 using any correct certification correctly computes  $\Gamma_n$  from  $\Gamma_{n-1}$ .*
  - 276 2. *Using the predicate  $\Xi$  as certification function, it runs in  $O(n\kappa_n)$  time and  $O(n)$  space.*
- 277 Moreover, it is worth noticing that Algorithm 1 is embarrassingly parallelizable.

### 278 3.3 Alternative certification function.

279 Let us propose a **second certification function** that derives from the rule based charac-  
280 terization of autocorrelations also presented in [8]. It states that a period set of  $\Gamma_n$  must  
281 i/ contains the trivial period zero, ii/ satisfy the Forward Propagation Rule (FPR), and  
282 iii/ the Backward Propagation Rule (BPR). The rule based characterization is shown to be  
283 equivalent to the predicate  $\Xi$  in Theorem 5.1 from [8] (see Appendix F.2).

284 The pseudo-code of the incremental algorithm for computing  $\Gamma_n$  using the rule based  
285 certification function is shown and explained in Algorithm 3 in Appendix B.

## 286 4 Constructive certification of a period set

287 Let  $R$  be subset of  $\{0, 1, \dots, n - 1\}$ . We say that a word  $u$  *realizes*  $R$  if  $P(u) = R$ . Another  
288 interesting certification function is: to attempt to build a word  $u$  that realizes  $R$ ; if the  
289 attempt succeeds,  $R$  is a valid period set. Given the alphabet independence of  $\Gamma_n$ , we restrict  
290 the search to binary strings.

291 Below we present an algorithm for the *binary realization* of a set (see Algorithm 2). Using  
292 it as a certification function in Algorithm 1, the latter will compute  $\Gamma_n$  from  $\Gamma_{n-1}$  and also  
293 yield one realizing string for each period set.

### 294 4.1 Binary realization of a subset of $\{0, 1, \dots, n - 1\}$

295 Algorithm 2 computes a word  $u$  that realizes a set  $P$  for length  $n > 0$ , or returns the empty  
296 word  $\epsilon$  if  $P$  is not a valid period set of  $\Gamma_n$ . For legibility, the preliminary checks on  $P$  are not  
297 written in Algorithm 2: they include checking that  $P$  is a subset of  $[0, \dots, n - 1]$ , is ordered,  
298 and has zero as first period. The word  $u$  is written over the alphabet  $\{\mathbf{a}, \mathbf{b}\}$ .

299 The algorithm considers elements of  $P$  backwards, starting with largest integer first, since  
300  $P$  is ordered. At each execution of the for loop, it considers the current integer  $P[i]$  as a  
301 period and builds a suffix of  $u$  of length  $n - P[i]$  (variable  $lg$ ). In fact, it considers a potential  
302 larger and larger nested sets, and computes a suffix of  $u$  for this length. At the end of the  
303 for loop, the variable *suffix* contains a string of length  $lg$  realizing the nested set. Note that  
304 algorithm uses three variables (whose names start with *prev*) to store the length, the inner  
305 period, and the suffix obtained with the previous period.

---

**Algorithm 2** Binary Realization
 

---

```

Input :  $n > 0$ : integer;  $P$ : a subset of  $[0, 1, \dots, n - 1]$  including 0, in a sorted array
Output : a binary string realizing  $P$  at length  $n$  xor the empty string otherwise;

1  $k := \text{card}(P)$  ; //  $k$ : cardinality of  $P$ 
2 if  $k = 1$  then return  $\mathbf{a.b}^{(n-1)}$  // trivial case where  $P = \{0\}$ ;
  // processing the largest period and init. variables
3  $\text{prevLg} := n - P[k - 1]$ ;  $\text{prevIP} := \text{prevLg}$ ;  $\text{prevSuffix} := \mathbf{a.b}^{(\text{prevLg}-1)}$ ;
4 for  $i$  going from  $k - 2$  to 0 do
5    $\text{lg} := n - P[i]$ ;
6    $\text{innerPeriod} := P[i + 1] - P[i]$ ;
7   if  $\text{innerPeriod} < \text{prevIP}$  then return  $\epsilon$  ;
8   if  $\text{lg} \leq 2 \times \text{prevLg}$  then // condition for case 1
9     if  $(\text{innerPeriod} = \text{prevIP})$  OR  $((\text{prevIP} \nmid \text{innerPeriod}) \text{ AND } ($ 
10       $(\text{innerPeriod} = \text{prevLg}) \text{ OR } (\text{prevSuffix} \text{ has period } \text{innerPeriod}))$  ) then
11      // suffix := a prefix of prevSuffix concat. with prevSuffix
12       $\text{suffix} := \text{prevSuffix}[0..\text{innerPeriod}-1] . \text{prevSuffix}$ ;
13    else return  $\epsilon$  // invalid case for length lg;
14  else // condition for case 2
15    // suffix := prevSuffix newsymbols prevSuffix
16     $\text{nb} := \text{lg} - 2 \times \text{prevLg}$ ;
17     $\text{newPrefix} := \text{prevSuffix} . \mathbf{a}^{\text{nb}}$ ;
18    if newPrefix is not primitive then
19       $\text{newPrefix} := \text{prevSuffix} . \mathbf{a}^{(\text{nb}-1)}\mathbf{b}$ ;
20    // Invariant: newPrefix is primitive
21     $\text{suffix} := \text{newPrefix} . \text{prevSuffix}$ ;
22    // update variables
23     $\text{prevLg} := \text{lg}$ ;  $\text{prevIP} := \text{innerPeriod}$ ;  $\text{prevSuffix} := \text{suffix}$ ;
24 return suffix;

```

---

306 The base case is processed before the loop and consider the nested set for  $P[k - 1] =$   
 307  $\text{max}(P)$  for the length  $n - \text{max}(P)$  without any period. Hence, the suffix  $\mathbf{a.b}^{(\text{prevLg}-1)}$  is a  
 308 realization for nested set  $\{0\}$  for length  $n - \text{max}(P)$ .

309 In the for loop the key variable is the *innerPeriod*, which equals the offset  $P[i + 1] - P[i]$ ,  
 310 which is the basic period of the current nested set. If  $\text{innerPeriod} < \text{prevIP}$  then the FPR  
 311 is violated and the algorithm returns  $\epsilon$  (line 7). Two cases are considered depending on  
 312 whether the current length is smaller twice the previous length (case 1) or not (case 2).  
 313 Because of the notion of period, the suffix must start and end with a copy of *prevSuffix*. The  
 314 construction of the suffix depends on the case. In case 1, the two copies of *prevSuffix* are  
 315 concatenated or overlap themselves, and some additional conditions are required (line 9).  
 316 These conditions are dictated by the characterization of period set from [8] (see the predicate  
 317  $\Xi$  in Appendix F.2). Whenever one is not satisfied, Algorithm 2 returns the empty word as  
 318 expected. In case 2, the two copies of *prevSuffix* must be separated by  $\text{nb}$  additional symbols  
 319 (to be determined). One builds a *newPrefix* that starts with *prevSuffix* followed by  $\mathbf{a}^{\text{nb}}$ , and  
 320 one checks whether *newPrefix* is primitive. This *newPrefix* is the part that ensures the suffix  
 321 will have *innerPeriod* as a period. The primitivity is required, since *newPrefix* may have a

## XX:10 Incremental algorithms for the set of period sets

322 proper period, but this period shall not divide *innerPeriod*. If primitivity is not satisfied,  
 323 then changing the last symbol of *newPrefix* by a **b** will make it primitive. This is enforced by  
 324 Lemma 3 from [10], which states that for any binary word  $w$ ,  $wa$  or  $wb$  is primitive. So we  
 325 know that at least one of the two forms of *newPrefix* is primitive as necessary. It can be  
 326 that both are primitive and suitable. Finally, we build the current *suffix* by concatenating  
 327 *newPrefix* with *prevSuffix*.

328 **Complexity** First, in case 1, checking the condition "*prevSuffix* has period *innerPeriod*"  
 329 can be done in linear time in  $|prevSuffix|$  (which is  $\leq n$ ). Overall, this can be executed  
 330  $card(P)$  times. Second, the primitivity test performed in case 2 takes a time proportional to  
 331 the length of the string *newPrefix*. However, the sum of these lengths, for all iterations of the  
 332 loop, is bounded by  $n$ . Other instructions of the for loop take constant time. Overall, the  
 333 time complexity of Algorithm 2 by  $O(card(P) \times n)$ . However, when Algorithm 2 is plugged  
 334 in Algorithm 1 it processes special instances: either  $P$  or  $Q := P \cup \{n-1\}$ , with  $P \in \Gamma_{n-1}$ .  
 335 Then, the time taken by all verifications of condition "*prevSuffix* has period *innerPeriod*" for  
 336 all cases 1 is bounded by  $n$ , due to the properties of periods that generate more than two  
 337 repetitions in a string (see Lemma 15 and Lemma 2 from [10]). For the instances processed  
 338 in Algorithm 1, the time complexity of Algorithm 2 is  $O(card(P) + n)$  or  $O(n)$ .

339 **Remark** It is possible to modify Algorithm 2 to build, instead of a binary word, a  
 340 realizing string that maximizes the number of distinct symbols used in it. Indeed, new  
 341 symbols are used only in the base case and in case 2. Each time, it is possible to choose  
 342 symbols that have not been used earlier in the algorithm, and thus to maximize the overall  
 343 number. Note that this would remove the need of the primitivity test in case 2.

### 344 4.2 Examples of binary realization

345 We take the case of a valid period set at length  $n = 9$  that becomes invalid at  $n = 10$ , and  
 346 show the traces of execution for both lengths. Let  $P = \{0, 3, 6, 8\}$  and first  $n = 9$ . The  
 347 operator  $\oplus_j$  merges the two strings with an offset of length  $j$  if the corresponding prefix  
 348 and suffix are equal, for any appropriate integer  $j$ . So when  $n = 9$ , the merge  $v = w \oplus_3 w$   
 349 with  $w = abaaba$  is feasible since  $w$  has period 3. When  $n = 10$ , the merge  $w = y \oplus_3 y$  with  
 350  $y = abab$  is not possible since  $a \neq b$ . The trace for  $n = 10$  is in Appendix C.

period	length	inner period	case	suffix	valid
8	$9-8 = 1$	$9-8 = 1$	2	$z = a$	true
6	$9-6 = 3$	$8-6 = 2$	2	$y = zbz = aba$	true
3	$9-3 = 6$	$6-3 = 3$	2	$w = yy = abaaba$	true
0	$9-0 = 9$	$3-0 = 3$	1	$v = w \oplus_3 w = (aba)^3$	true

## 352 5 Fate and dynamics of period sets

353 One interest of incremental algorithms is to shed light on the dynamics of the  $\Gamma_n$  when  $n$   
 354 increases, both in terms of new and dying period sets, as well as on the structure of  $\Gamma_n$ . As  
 355 mentioned in introduction,  $\Gamma_n$  is a lattice under inclusion; the union and the intersection  
 356 of two period sets are period sets [24]. Even if the cardinality of  $\Gamma_n$  increases with  $n$ , the  
 357 growth is not regular. It is worth investigating the local dynamics of  $\Gamma_n$  when  $n$  changes,  
 358 and for this we define the fate of period sets.

## 5.1 Fate of a period set when $n$ increases

The incremental algorithms presented above show that  $\Gamma_{n-1}$  and  $\Gamma_n$  share some period sets, and that other period sets are derived by an extension, that is are of the form  $P \cup \{n-1\}$ .

Let  $P$  be a period set of  $\Gamma_{n-1}$ . The maximal period in  $P$ , denoted  $\max(P)$ , determines the first length at which  $P$  exists: indeed, the *birth* of  $P$  occurs in  $\Gamma_{\max(P)+1}$ . When the length increases, say from  $n-1$  to  $n$ , what can be the fate of  $P$ ? Only three possibilities exist:

1. either  $P$  remains valid at length  $n$ ,
2. or  $P$  has an *extension* with period  $n-1$  at length  $n$ ,
3. or  $P$  *dies* (i.e., is neither case 1 nor case 2), see Definition 14 in Appendix B.

Note that cases 1 and 2 are not exclusive from each other. Let us illustrate these with the following example.

► **Example 13.** For instance,  $\{0, 3, 6\}$  is born in  $\Gamma_7$  and also belongs to  $\Gamma_8$  and  $\Gamma_9$ ; its extension  $\{0, 3, 6, 7\}$  also belongs to  $\Gamma_8$ . This extension is not compulsory since  $\{0, 3, 6\}$  also belongs to  $\Gamma_8$ . From a dynamic view point, one can say that  $\{0, 3, 6\}$  from  $\Gamma_7$  *generates* both  $\{0, 3, 6\}$  and  $\{0, 3, 6, 7\}$  in  $\Gamma_8$ . On the contrary,  $\{0, 4, 6\}$  belongs  $\Gamma_7$ , but dies at  $n=8$ , since the pair of periods  $(4, 6)$  satisfies the FW condition at that length and would require to add  $\gcd(4, 6)$  as a new basic period. Last,  $\{0, 2, 4, 6\}$  belongs to  $\Gamma_7$ ,  $\Gamma_8$ , and generates  $\{0, 2, 4, 6, 8\}$  in  $\Gamma_9$  because the extension with period 8 is required by the FPR.

The fate of depends on

1. the smallest length at which an extension will be required (i.e., as a consequence of the FPR a new period is added to  $P$  at some length); we call it the *extension limit* of  $P$ ,
2. the smallest length at which some pairs of periods violates the BPR (we call it the *Recursive FW limit*).

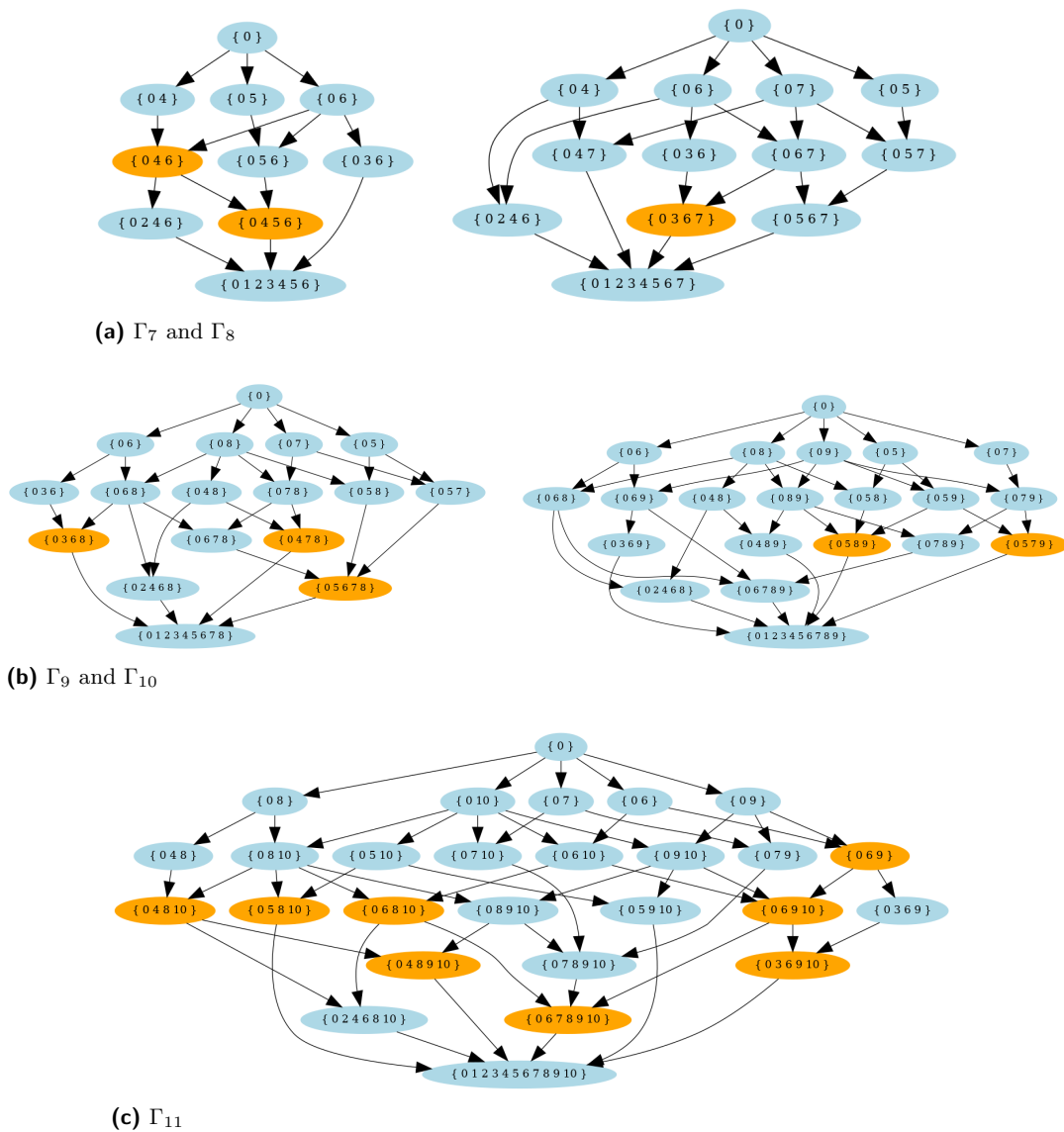
These two lengths depend only on the period set, and can thus be computed as soon as  $P$  is born. We provide in Appendix E two algorithms to compute these limits.

Figure 1 shows for  $n = 7, \dots, 11$ , the set  $\Gamma_n$  represented as a lattice of period sets with the inclusion relationship. The out-going arrows of a period set point to its successors in the lattice. The dying period sets of  $\Gamma_n$ , that is those that do not exist at length  $n+1$ , are shown in orange background. The number of dying period sets is not monotonically increasing: it equals 2 in  $\Gamma_7$ , 1 in  $\Gamma_8$ , 3 in  $\Gamma_9$ , 2 in  $\Gamma_{10}$ , and 8 in  $\Gamma_{11}$ , for instance. Clearly, it does not prevent the cardinality of  $\Gamma_n$  to increase monotonically. An interesting question for future work is to find the function that for  $n$  gives the number of dying period sets of  $\Gamma_n$ , and how these are distributed with respect to their basic period.

## 6 Conclusion and exploration of $\Gamma_n$ : distribution of period sets with respect to basic period and weight

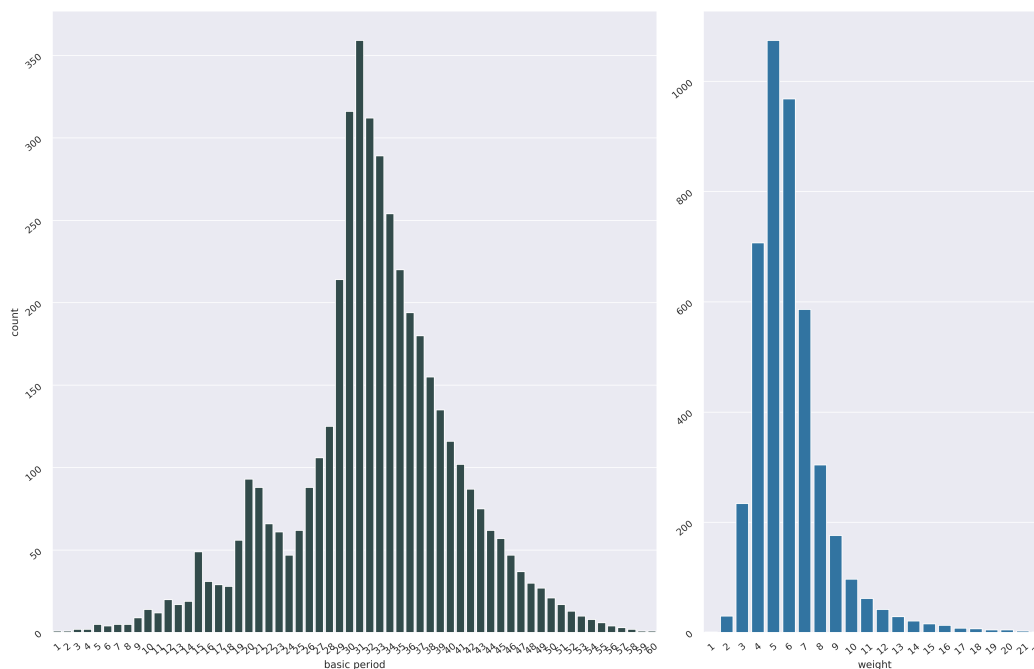
The key element of a period set is its basic period, which defines the first level of periodicity in a word. How period sets in  $\Gamma_n$  are distributed according to their basic period is non trivial. Enumerating  $\Gamma_n$  allows inspecting this distribution. The left plot in Figure 2 displays  $\kappa_{n,p}$ , the counts of period sets for all possible basic periods  $p$ , in  $\Gamma_{60}$ . In predicate  $\Xi$  [8], as well as in the dynamic programming algorithm that enumerates  $\Gamma_n$  [24], one separates period sets depending on the basic period being larger than  $\lfloor n/2 \rfloor$  (case **b**) or smaller than or equal to it (case **a**). The smooth decrease of counts beyond the basic period equals to half of the string length is explained by the combinatorial property that links number of period sets in case **b** and the number of binary partitions of an integer (see Lemma 5.8 in [24]). However,

**XX:12** Incremental algorithms for the set of period sets



■ **Figure 1** Lattices representations of  $\Gamma_7$  and  $\Gamma_8$  (a),  $\Gamma_9$  and  $\Gamma_{10}$  (b), and of  $\Gamma_{11}$  (c). Each node contains a period set (a list of periods separated by spaces). Those whose last period equals  $(n - 1)$  are obtained by extension of period set from  $\Gamma_{n-1}$ , and those nodes in orange background are dying period sets at length  $n + 1$ .

404 the distribution of counts for all period sets in case **a**, still requires some investigation and  
 405 statistical modeling. Here, we observe that between basic period 1 and 30,  $\kappa_{n,p}$  increases  
 406 globally with the basic period  $p$ , but locally  $\kappa_{n,p}$  increases and then decreases to reach local  
 407 maxima when  $p$  divides the string length  $n$  (e.g. see the peaks at  $p = 10, 12, 15, 20, 30$ , which  
 408 correspond to period sets of case **a**).



■ **Figure 2** Distribution in  $\Gamma_{60}$  of the number of period sets by basic period (left) and by weight (right), for string length of  $n := 60$ . Beyond basic period 30, the counts decrease smoothly with the basic period. Between basic period 1 and 30 the counts increase to a local maximum when the basic period reaches  $\lfloor n/x \rfloor$  for  $1 < x \leq 12 =$  (e.g. basic periods 10, 12, 15, 20, 30). The distribution by weight (right) is limited to weight below 22; it is unimodal and right skewed towards low weights.

409 Other works have investigated combinatorial parameters that control the number of  
 410 periods of a word [7]. Thanks to enumeration of  $\Gamma_n$  one can study the real distribution of  
 411 weight of period sets and how it evolves with  $n$ . The right plot of Figure 2 displays the  
 412 number of period sets having the same weight (i.e. same number of periods) for  $n = 60$ .  
 413 This distribution is right skewed and illustrates the constraints imposed by multiple periods.  
 414 Similar figures for other string lengths are shown in Appendix A.

415 **Conclusion.** We provide algorithms to enumerate  $\Gamma_n$  incrementally with low space  
 416 requirement, and an algorithm for binary realization of a period set. They allow to inspect  
 417  $\Gamma_n$  and to visualize how parameters like the weight or the basic period impact the number of  
 418 PS. We define the fate of a PS and propose to study the dynamics of  $\Gamma_n$  when  $n$  increases.  
 419 Many questions remain: how can the recursive FW and extension limits of PS of  $\Gamma_{n-1}$  be  
 420 used to speed up the incremental enumeration of  $\Gamma_n$ ? Can we exploit binary realizing strings  
 421 to ease enumeration or to unravel how population sizes evolve with  $n$ ? Among directions for  
 422 future work, finding algorithms to enumerate PS for generalizations of words, like partial  
 423 words or multidimensional words (aka matrices) is interesting. As seen in Figure 1, the  
 424 number of PS that die in function of  $n$  is not monotonically increasing; thus understanding  
 425 the sequences of  $\kappa_n$  and  $\kappa_{n,p}$  is both stimulating and challenging (see also Figures 2, 3-5).

## 426 — References

- 427 1 Dragana Bajic and Tatjana Loncar-Turukalo. A simple suboptimal construction of cross-  
428 bifix-free codes. *Cryptography and Communications*, 6(6):27–37, 2014. doi:10.1007/  
429 s12095-013-0088-8.
- 430 2 Stefano Bilotta, Elisa Pergola, and Renzo Pinzani. A new approach to cross-bifix-free sets. *IEEE*  
431 *Transactions on Information Theory*, 58(6):4058–4063, 2012. doi:10.1109/TIT.2012.2189479.
- 432 3 Francine Blanchet-Sadri, Justin Fowler, Joshua D. Gafni, and Kevin H. Wilson. Combinatorics  
433 on partial word correlations. *J. Comb. Theory, Ser. A*, 117(6):607–624, 2010. doi:10.1016/j.  
434 jcta.2010.03.001.
- 435 4 M.Gabriella Castelli, Filippo Mignosi, and Antonio Restivo. Fine and wilf’s theorem  
436 for three periods and a generalization of sturmian words. *Theoretical Computer Sci-*  
437 *ence*, 218(1):83–94, April 1999. URL: [http://dx.doi.org/10.1016/S0304-3975\(98\)00251-5](http://dx.doi.org/10.1016/S0304-3975(98)00251-5),  
438 doi:10.1016/s0304-3975(98)00251-5.
- 439 5 Andrzej Ehrenfeucht and D.M. Silberger. Periodicity and unbordered segments of words. *Dis-*  
440 *crete Mathematics*, 26(2):101–109, 1979. URL: [http://dx.doi.org/10.1016/0012-365X\(79\)](http://dx.doi.org/10.1016/0012-365X(79)90116-X)  
441 [90116-X](http://dx.doi.org/10.1016/0012-365X(79)90116-x), doi:10.1016/0012-365x(79)90116-x.
- 442 6 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math.*  
443 *Soc.*, 16:109–114, 1965.
- 444 7 Daniel Gabric, Narad Rampersad, and Jeffrey Shallit. An inequality for the number of periods  
445 in a word. *International Journal of Foundations of Computer Science*, 32(05):597–614, Jun  
446 2021. doi:10.1142/s0129054121410094.
- 447 8 Leo J. Guibas and Andrew M. Odlyzko. Periods in strings. *J. of Combinatorial Theory series*  
448 *A*, 30:19–42, 1981. doi:10.1016/0097-3165(81)90038-8.
- 449 9 Dan Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.
- 450 10 Vesa Halava, Tero Harju, and Lucian Ilie. Periods and binary words. *J. Comb. Theory, Ser. A*,  
451 89(2):298–303, 2000. URL: <https://doi.org/10.1006/jcta.1999.3014>, doi:10.1006/JCTA.  
452 1999.3014.
- 453 11 Stepan Holub and Jeffrey O. Shallit. Periods and borders of random words. In Nicolas Ollinger  
454 and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science,*  
455 *STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 44:1–44:10.  
456 Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.STACS.2016.  
457 44.
- 458 12 D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of*  
459 *Computing*, 6:323–350, 1977.
- 460 13 Douglas A. Leonard. A prefix problem. *Ars Combinatoria*, 24:51–55, 1987.
- 461 14 Paul Leopardi. Testing the Tests: Using Random Number Generators to Improve Empirical  
462 Tests. In Pierre L’Ecuyer and Art B. Owen, editors, *Monte Carlo and Quasi-Monte Carlo*  
463 *Methods 2008*, pages 501–512. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-  
464 04107-5\_32. URL: [http://link.springer.com/chapter/10.1007/978-3-642-04107-5\\_32](http://link.springer.com/chapter/10.1007/978-3-642-04107-5_32).
- 465 15 M. Lothaire, editor. *Combinatorics on Words*. Cambridge University Press, second edition,  
466 1997.
- 467 16 Veli Mäkinen, Djamel Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale*  
468 *Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*.  
469 Cambridge University Press, 2015. doi:10.1017/CB09781139940023.
- 470 17 G. Marsaglia and A. Zaman. Monkey tests for random number generators. *Computers and*  
471 *Mathematics with Applications*, 26(9):1–10, 1993.
- 472 18 P. Nielsen. A note on bifix-free sequences (corresp.). *IEEE Transactions on Information Theory*,  
473 19(5):704–706, September 1973. URL: <http://dx.doi.org/10.1109/TIT.1973.1055065>, doi:  
474 10.1109/tit.1973.1055065.
- 475 19 P. Nielsen. On the expected duration of a search for a fixed pattern in random data (corresp.).  
476 *IEEE Transactions on Information Theory*, 19(5):702–704, September 1973. URL: <http://dx.doi.org/10.1109/TIT.1973.1055064>, doi:10.1109/tit.1973.1055064.  
477

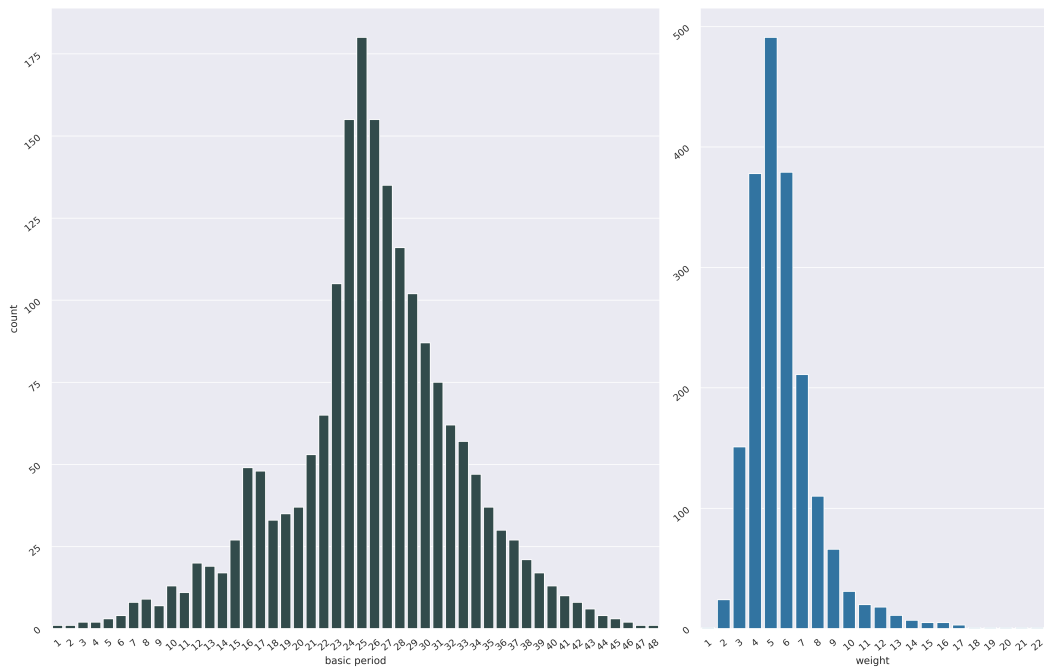


- 478 20 Ora E. Percus and Paula A. Whitlock. Theory and Application of Marsaglia’s Monkey Test for  
479 Pseudorandom Number Generators. *ACM Transactions on Modeling and Computer Simulation*,  
480 5(2):87–100, April 1995.
- 481 21 Sven Rahmann and Eric Rivals. Exact and efficient computation of the expected number  
482 of missing and common words in random texts. In *Proc. of CPM 2000*, page 375–387.  
483 Springer Berlin Heidelberg, 2000. URL: [http://dx.doi.org/10.1007/3-540-45123-4\\_31](http://dx.doi.org/10.1007/3-540-45123-4_31),  
484 doi:10.1007/3-540-45123-4\_31.
- 485 22 Sven Rahmann and Eric Rivals. On the distribution of the number of missing words in  
486 random texts. *Combinatorics, Probability and Computing*, 12(01), Jan 2003. URL: <http://dx.doi.org/10.1017/S0963548302005473>, doi:10.1017/s0963548302005473.
- 488 23 Eric Rivals and Sven Rahmann. Combinatorics of periods in strings. In *Proc. of ICALP*  
489 *2001*, page 615–626. Springer Berlin Heidelberg, 2001. URL: [http://dx.doi.org/10.1007/3-540-48224-5\\_51](http://dx.doi.org/10.1007/3-540-48224-5_51), doi:10.1007/3-540-48224-5\_51.
- 491 24 Eric Rivals and Sven Rahmann. Combinatorics of periods in strings. *Journal of Combinatorial Theory, Series A*, 104(1):95–113, Oct 2003. URL: [http://dx.doi.org/10.1016/S0097-3165\(03\)00123-7](http://dx.doi.org/10.1016/S0097-3165(03)00123-7), doi:10.1016/s0097-3165(03)00123-7.
- 494 25 Eric Rivals, Michelle Sweering, and Pengfei Wang. Convergence of the Number of Period Sets  
495 in Strings. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International  
496 Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz  
497 International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:14, Dagstuhl, Germany,  
498 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/opus/volltexte/2023/18152>, doi:<https://doi.org/10.4230/LIPIcs.ICALP.2023.100>.
- 500 26 S. Robin and J.-J. Daudin. Exact distribution of word occurrences in a random sequence of  
501 letters. *J. of Applied Probability*, 36:179–193, 1999.
- 502 27 Stéphane Robin, François Rodolphe, and Sophie Schbath. *DNA, Words and Models*. Cambridge  
503 Univ. Press, 2005.
- 504 28 William F. Smyth. *Computing Pattern in Strings*. Pearson - Addison Wesley, 2003.
- 505 29 R. Tijdeman and L. Zamboni. Fine and wilf words for any periods. *Indagationes Mathematicae*,  
506 14(1):135–147, March 2003. URL: [http://dx.doi.org/10.1016/S0019-3577\(03\)90076-0](http://dx.doi.org/10.1016/S0019-3577(03)90076-0),  
507 doi:10.1016/s0019-3577(03)90076-0.
- 508 30 R. Tijdeman and L.Q. Zamboni. Fine and wilf words for any periods ii. *Theoretical Computer  
509 Science*, 410(30–32):3027–3034, August 2009. URL: <http://dx.doi.org/10.1016/j.tcs.2009.02.004>,  
510 doi:10.1016/j.tcs.2009.02.004.
- 511 31 E. Ukkonen. Approximate string-matching with  $q$ -grams and maximal matches. *Theor. Comp.  
512 Sci.*, 92(1):191–211, January 1992.

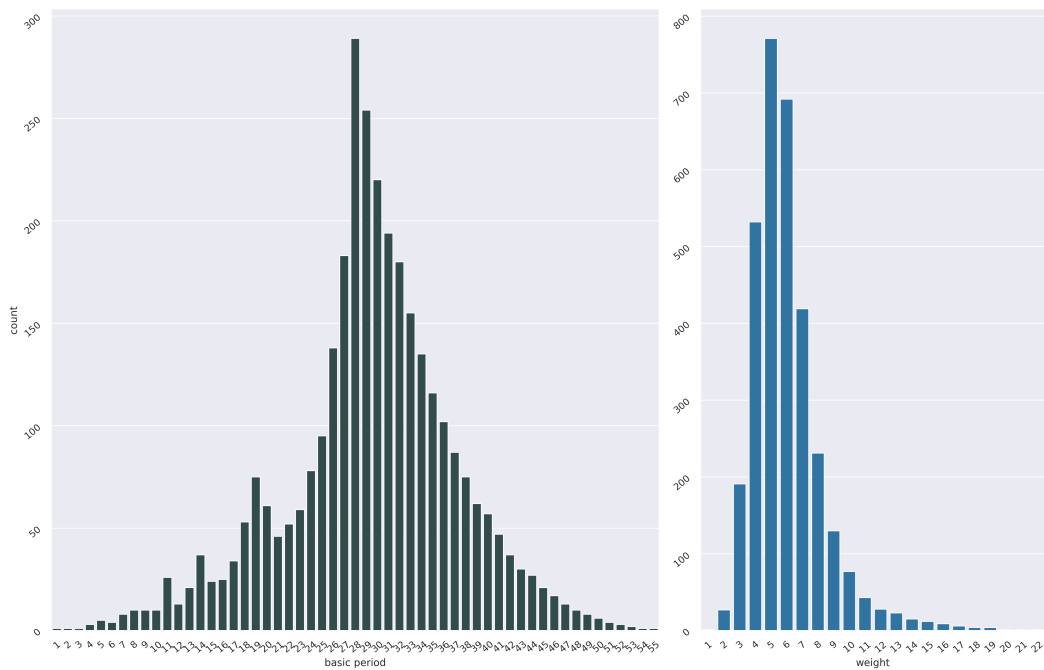
## 513 **A** Exploration of $\Gamma_n$ : Distribution of the number of period sets by 514 basic period and by weight

515 Like in Figure 2, we explore how period sets are distributed according to their basic period,  
516 and according to their weight for other string lengths. We plot these distributions for  $n = 48$ ,  
517  $n = 55$ , and  $n = 59$  in Figures 3, 4, and 5, respectively. We choose these values because they  
518 differ in their number of divisors  $48 = 2^4 \times 3$ ,  $55 = 5 \times 11$  and 59 is prime. In essence, both  
519 plots for  $\Gamma_{48}$ ,  $\Gamma_{55}$ , and  $\Gamma_{59}$  look very similar to those for  $\Gamma_{60}$ . Even for a prime string length,  
520  $n = 59$ , the distribution of number of period sets in case **a**, shows a maximum at  $\lfloor n/2 \rfloor$  and  
521 local maxima at  $\lfloor n/3 \rfloor$ ,  $\lfloor n/4 \rfloor$  etc.

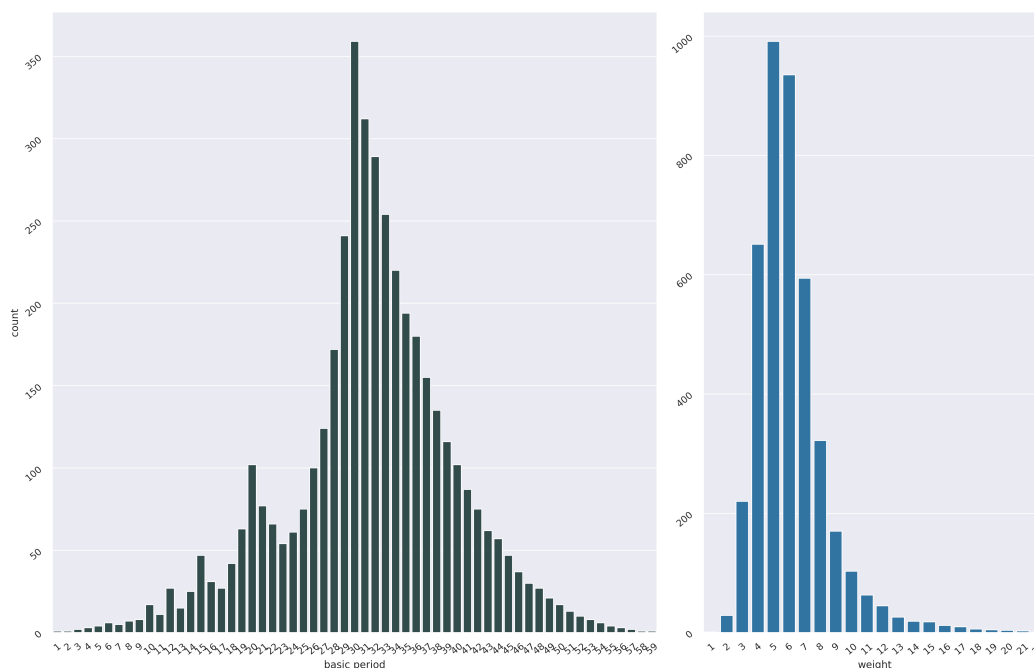
**XX:16** Incremental algorithms for the set of period sets



■ **Figure 3** Distribution in  $\Gamma_{48}$  of the number of period sets by basic period (left) and by weight (right), i.e., for string length of  $n := 48$ .



■ **Figure 4** Distribution in  $\Gamma_{55}$  of the number of period sets by basic period (left) and by weight (right), i.e., for string length of  $n := 55$ .



■ **Figure 5** Distribution in  $\Gamma_{59}$  of the number of period sets by basic period (left) and by weight (right), i.e., for string length of  $n := 59$ .

## 522 B Incremental algorithm with rule based certification.

523 Here, we detail an alternative version of the incremental algorithm, which uses the rule  
 524 based certification function derived from Theorem 5.1 from [8] (see also below). This is  
 525 related to subsection 3.3. Algorithm 3 presents the pseudo-code; it uses two functions named  
 526 *checkFPR* and *checkBPR*, which check if a set of integers satisfies respectively, the Forward  
 527 and Backward Propagation Rules.

528 In our case, as the candidate sets include a period set of  $\Gamma_{n-1}$ , they necessarily satisfy  
 529 the first condition (i). Regarding the FPR, since  $P$  belongs to  $\Gamma_{n-1}$ ,  $P$  satisfies the FPR  
 530 up to position  $n - 2$  included; and thus, only period  $n - 1$  can be required by the FPR. For  
 531 each possible pair  $(p, q)$  considered in the FPR, we only need to check if the FPR formula  
 532 yields  $(n - 1)$ . Second, for the same reason, when considering the candidate set  $P \cup \{n - 1\}$ ,  
 533 we are sure that the FPR is satisfied.

534 Let  $R$  be any subset of  $\{0, 1, \dots, n - 1\}$  containing zero and assuming that  $R$  is sorted in  
 535 increasing order, then we have that checking the FPR and BPR takes at most  $O(n \log_2(n))$   
 536 time (see Section 2).

537 Algorithm 3 differs from Algorithm 1 in two aspects. First, it can indicate for which  
 538 reason the candidate set is not a valid period set if the check fails. Second, it also computes  
 539 the set of "dying" period sets of  $\Gamma_{n-1}$ , that is the period set that do not remain valid at  
 540 length  $n$ , nor cannot be extended at length  $n$ . We will define these notions in Section 5.  
 541 Of course, dying period sets could also be computed within Algorithm 1, which uses the  
 542 predicate  $\Xi$  (but for simplicity and to avoid redundancy, was not mentioned earlier).

543 Altogether the time complexity of Algorithm 3 is bounded by  $O(n \log_2(n) \times \kappa_n)$ , which  
 544 may not be optimal.

## XX:18 Incremental algorithms for the set of period sets

545 ► **Definition 14.** A dying period set  $P$  is a period set of  $\Gamma_{n-1}$  such that neither  $P$  nor  
546  $P \cup \{n-1\}$  belong to  $\Gamma_n$ . In other words,  $P$  has no extension in  $\Gamma_n$ .

■ **Algorithm 3** IncrementalGamma with rule based certification

---

```
Input :  $n > 1$ : integer;  $\Gamma_{n-1}$ : the set of period sets for length  $n - 1$ 
Output:  $\Gamma_n$ : the set of period sets for length  $n$ ;  $D$ : the set of dying PS at length  $n$ ;
1  $G := \emptyset$ ; //  $G$ : variable to store  $\Gamma_n$ 
2  $D := \emptyset$ ; //  $D$ : variable to store dying PS
3 for  $P \in \Gamma_{n-1}$  do
4    $Q := P \cup \{n-1\}$  // build extension of  $P$  with period  $n-1$ ;
5   if  $checkFPR(P, n)$  then //  $n-1$  is required by FPR at length  $n$ 
6     if  $checkBPR(Q, n)$  then insert  $Q$  in  $G$ ;
7     else insert  $P$  in  $D$  // otherwise  $P$  is dying at length  $n$ ;
8   else
9     if  $checkBPR(P, n)$  then insert  $P$  in  $G$ ;
10    else
11      if  $checkBPR(Q, n)$  then insert  $Q$  in  $G$ ;
12      else insert  $P$  in  $D$  // otherwise  $P$  is dying at length  $n$ ;
13 return  $G$  and  $D$ ;
```

---

## 548 C Algorithm Binary realization

### 549 C.1 Correctness and complexity of the algorithm

550 **Proof.** Let us prove that the Algorithm Binary Realization is correct.

551 **Correction of the base case** As we process the last period of  $P$ , the nested set is  $\{0\}$   
552 for length  $n - \max(P)$ . We must build a suffix without period (i.e., whose basic period is its  
553 length). Hence, the word  $a.b^{(\text{prevLg}-1)}$  is a binary realization for this set.

554 **Correction of the general case.** After setting variables  $lg$  and  $innerPeriod$ , we check  
555 the condition ( $innerPeriod < prevIP$ ). In a period set, the offset  $P[i+1] - P[i]$  decreases  
556 when  $i$  increases. The condition implies the current nested set is invalid, and we return  $\epsilon$  as  
557 needed. Another way to formulate this: If the condition is satisfied, then  $suffix$ , which ends  
558 with  $prevSuffix$ , does not satisfy the FPR, meaning that this set is invalid.

559 The invariant at the start of the for loop is that  $prevSuffix$  realizes the nested set  $P_{P[i+1]}$   
560 and has  $prevIP$  as basic period. By construction, we know that  $lg = prevLg + innerPeriod$ .  
561 By construction,  $suffix$  ends with  $prevSuffix$  and has basic period  $innerPeriod$ . Thus, by the  
562 invariant,  $suffix$  will realize  $P_{P[i]}$ .

563 **Case 1** We build  $suffix$  by concatenating a prefix of  $prevSuffix$  of length  $innerPeriod$   
564 with  $prevSuffix$  (line 10), and we must ensure that  $suffix$  has basic period  $innerPeriod$ . Let  
565 us consider the conditions from line 9.

566 1. If ( $innerPeriod = prevIP$ ) then, as  $prevSuffix$  already has period  $prevIP$ ,  $suffix$  will  
567 inherit from it. Otherwise we know that ( $innerPeriod > prevIP$ ).

568 2. Then,  $prevSuffix$  has a basic period ( $prevIP$ ) that should not divide  $innerPeriod$ , which is  
569 the length of the prefix of  $prevSuffix$  that occurs as prefix of  $suffix$ . Hence, we require  
570 the condition ( $prevIP \nmid innerPeriod$ ) to be satisfied. Otherwise,  $suffix$  would also have  
571  $prevIP$  as period; then  $suffix$  would be a binary world, but would not realize  $P$ .

- 572 3. Then, if ( $innerPeriod = prevLg$ ) then  $lg = 2 \times prevLg$  and  $suffix$  equals  $/prevSuffix/2$  and  
 573 has the desired length and basic period.  
 574 4. Otherwise, we check that  $prevSuffix$  has period  $innerPeriod$ . If yes, then  $suffix$  also has  
 575 period  $innerPeriod$  by construction (line 10), and thus realizes  $P_{P[i]}$ . If not, then there is  
 576 no possible realization of  $P$  and we return  $\epsilon$  (line 11).

577 **Case 2** Here, we know that  $lg$  is larger than twice  $prevLg$ . Therefore, we will build a  
 578 prefix that starts with  $prevSuffix$  followed by  $nb$  new symbols, such that  $suffix$  has no period  
 579 shorter than  $innerPeriod$ . Hence, we must ensure that  $newPrefix$  is primitive, otherwise it  
 580 would have a period that divides  $innerPeriod$ . By Lemma 3 from [10], for any binary word  $w$ ,  
 581  $wa$  or  $wb$  is primitive. So, we concatenate  $a^{nb}$  to  $prevSuffix$ , and check if it is primitive (in  
 582  $O(|newPrefix|)$  time). If not, we change its last symbol by a  $b$ . In both cases,  $newPrefix$  is  
 583 primitive. By construction,  $suffix$  has basic period  $innerPeriod$  as desired, and thus realizes  
 584  $P_{P[i]}$ . ◀

### 585 C.2 Examples of traces of binary realizations

586 Here is the trace of Algorithm 2 for length  $n = 10$ , and  $P = \{0, 3, 6, 8\}$ , which is not a valid  
 587 period set for that length.

period	length	inner period	case	suffix	valid
8	10-8 = 2	10-8 = 2	2	$z = ab$	true
6	10-6 = 4	8-6 = 2	2	$y = zz = abab$	true
3	10-3 = 7	6-3 = 3	1	$w = y \oplus_3 y$	false

589 The table below illustrates that the merge attempted at the last loop iteration for  $P[i] = 3$   
 590 is impossible, since a mismatch occurs in the overlap.

pos.	0	1	2	3	4	5	6
$y$	a	b	a	b	-	-	-
$y$	-	-	-	a	b	a	b

### 592 D Checking FPR

593 Let us state some properties:

- 594 1. From the definition of FPR, we can see that checking the FPR for a pair  $(p, q)$  of  $P$  is  
 595 equivalent to checking the FPR for pair  $(0, q)$  in the nested PS  $P_p$ .
- 596 2. Assume the FPR is satisfied for pair  $(0, p)$ . Then, it is also satisfied for any pair  $(hp, jp)$   
 597 with  $1 \leq h < j < \lfloor n/p \rfloor$  and  $hp, jp \in P$ , since both periods are multiples of  $p$ .

598 From both properties, we get that once the FPR has been checked for the first pair  $(p, q)$   
 599 taken that has offset  $(q - p)$ , it is also satisfied for any other pair whose offset equals  $r$  or a  
 600 multiple of  $r$ . It follows that, for a set  $P$ , one can limit the checking of FPR only to left most  
 601 pairs whose offsets differ from each other and are not multiple of another offset. Thus, at least  
 602 one element, say  $p$ , must be an *irreducible period* (as defined in [24]), and  $q$  is the closest  
 603 period to  $p$  (i.e., one which gives rise to the smallest offset with respect to  $p$ ). Since, the  
 604 number of irreducible periods of a period set of  $\Gamma_n$  is bounded by  $\log_2(n)$  [25], the number  
 605 of such pairs also is. We obtain the bound on the complexity for the general case stated in  
 606 Lemma 8.

607 **E Fate: computation of the limits of a period set**

608 **E.1 Extension limit**

609 Algorithm 4 computes the extension limit of  $P$ . The extension limit is a length at which some  
 610 deducible period needs to be added to  $P$  to satisfy the FPR. It equals the added period plus  
 611 one, and must be larger than the birth length of  $P$  (Indeed,  $P$  is a valid period for length at  
 612 which is first occurs, and thus satisfies the FPR for that length). By definition of the FPR, a  
 613 period induced by the FPR equals  $P[i] + P[i] - P[j]$  for some indexes  $0 < j < i < \text{card}(P)$ .  
 614 Because, we need the minimum of added periods, we can restrict the computation to pairs  
 615 of adjacent periods (i.e. that is to case where  $j = i - 1$ ), since the offset between periods  
 616 decreases with their index. Hence, the formula  $P[i] + (P[i] - P[i - 1])$  for computing the  
 617 limit induced from current period  $P[i]$ . Because of this, we can also rule out cases where  
 618  $P[i]$  is smaller the half the birth length of  $P$  (line 6).

■ **Algorithm 4** ExtensionLimit

---

**Input** :  $P$ : a valid period set (as an ordered list of integers)  
**Output**: the extension limit of  $P$  (a minimum length at which  $P$  requires an extension);

```

1 birthLg := max(P) + 1; // min length x at which P first occurs in Γ(x)
2 limit := max(int); // limit to be computed, init. with largest integer
3 for i := card(P) - 1 to 1 do
619 4 | if P[i] ≤ ⌊  $\frac{\text{birthLg}}{2}$  ⌋ then //
5 | | break; // avoid such P[i] values whose limit cannot be > birthLg
6 | if P[i] + (P[i] - P[i - 1]) ≥ birthLg then // current limit is beyond
| | birthLg
| | // update limit with the min of limit and current limit
7 | | limit := min(limit, P[i] + (P[i] - P[i - 1]));
8 return limit + 1;
```

---

620 **E.2 Recursive FW limit**

621 We exhibit an algorithm to compute what we termed, the recursive FW limit of a PS  $P$  (see  
 622 Algorithm 5). The FW theorem provides a way to compute a maximal length for any pair  
 623 of distinct, non trivial periods such that one period is not a multiple of the other. For any  
 624  $p, q$  in  $P$  such that  $0 < p < q < n$  and  $p \nmid q$ , we denote by  $FW(p, q)$  the FW limit, that is  
 625  $FW(p, q) := p + q - \text{gcd}(p, q)$ . If  $p \div q$  we assume that  $FW(p, q) := \text{max}(int)$ . First, the  
 626 algorithm proceeds with two special cases: if all periods are multiple of the basic period,  
 627 then it returns  $\text{max}(int)$ . Note this includes the case with basic period equals to one. If  $P$   
 628 contains only three periods, then it returns  $FW(P[1], P[2])$ .

629 Otherwise, it will compute the limit  $l$  and initializes with  $\text{max}(int)$ . It loops over  $P$   
 630 backwards, to consider longer and longer suffixes starting at a position with period of a word  
 631 satisfying  $P$ , and builds a list  $Q$  of periods restricted to the current suffix. The periods in  
 632  $Q$  are those of  $P$  minus the starting position. It computes  $FW(Q[1], Q[2])$  and takes the  
 633 minimum between  $l$  and  $P[i] + FW(Q[1], Q[2])$ . After terminating the loop, it returns the  
 634 limit  $l$ .

■ **Algorithm 5** RecursiveFWLimit

---

**Input** :  $P$ : a valid period set (as an ordered list of integers)  
**Output**: the minimum length at which a pair of periods of  $P$  requires a change of basic period (application of FW theorem);

```

1 if ( $P[1] \mid P[i]$ ) for all  $1 < i < \text{card}(P)$  then // If basic period divides all
  other periods
2   return  $\max(\text{int})$ ;
3 if  $\text{card}(P) = 3$  then // If  $P$  contains only two non trivial periods
4   return  $\text{FW}(P[1], P[2])$ ;
635
5 limit :=  $\max(\text{int})$ ; // limit to be computed, init. with largest integer
6 insert ( $P[n-1] - P[n-2]$ ) in  $Q$ ; // Init  $Q$  with the last offset between
  periods
7 for  $i := \text{card}(P) - 3$  to 0 do
8   offset :=  $P[i+1] - P[i]$ ;
9    $Q[0] := Q[0] + \text{offset}$ ;
10  insert offset at first position in  $Q$ ;
11  limit :=  $\min(\text{limit}, P[i] + \text{FW}(Q[0], Q[1]))$ ;
12 return limit;
```

---

636 **Complexity.** In Algorithm 5, the first special case is processed in  $\text{card}(P)$  time (lines  
637 1–2), while the second one requires constant time (lines 3–4). The main loop is executed at  
638 most  $\text{card}(P)$  times and all instructions in it take constant time (lines 7–11). Altogether,  
639 Algorithm 5 takes  $O(\text{card}(P))$  time and constant space.

640 **Correctness.** The correctness of Algorithm 5 follows from Lemma 7.

641 **F Properties of periods and characterization of period sets**

642 **F.1 Properties of periods**

643 Let us state some known, useful properties of periods, which are detailed in [25].

644 ► **Lemma 15.** Let  $p$  be a period of  $u \in \Sigma^n$  and  $k \in \mathbb{N}_{\geq 0}$  such that  $kp < n$ . Then  $kp$  is also a  
645 period of  $u$ .

646 ► **Lemma 16.** Let  $p$  be a period of  $u \in \Sigma^n$  and  $q$  a period of the suffix  $w = u[p..n-1]$ .  
647 Then  $(p+q)$  is a period of  $u$ . Moreover,  $(p+kq)$  is also a period of  $u$  for all  $k \in \mathbb{N}_{\geq 0}$  with  
648  $p+kq < n$ .

649 ► **Lemma 17.** Let  $p, q$  be periods of  $u \in \Sigma^n$  with  $0 \leq q \leq p$ . Then the prefix and the suffix  
650 of length  $(n-q)$  have the period  $(p-q)$ .

651 ► **Lemma 18.** Suppose  $p$  is a period of  $u \in \Sigma^n$  and there exists a substring  $v$  of  $u$  of length  
652 at least  $p$  and with period  $r$ , where  $r \mid p$ . Then  $r$  is also a period of  $u$ .

653 **F.2 Characterization of autocorrelations/period sets [8]**

654 Guibas and Odlyzko have provided two equivalent characterizations of period sets: one  
655 is given by predicate  $\Xi$ , the other is the rule based characterization given in Section 2.3.  
656 However, they manipulate period sets as binary vectors called *autocorrelation* (or sometimes  
657 correlation for short). Remind that an autocorrelation is a binary encoding in a binary



## XX:22 Incremental algorithms for the set of period sets

658 string of length  $n$  of a period set of  $\Gamma_n$ . We recall in extenso the original predicate  $\Xi$  and  
659 then their Theorem 5.1, which states the equivalence of characterizations and the alphabet  
660 independence.

*Predicate  $\Xi$ :*  $v$  satisfies  $\Xi$  iff  $v_0 = 1$  and, if  $p$  is the basic period of  $v$ , one of the following conditions is satisfied:

*Case a:*  $p \leq \lfloor n/2 \rfloor$ . Let  $r := \text{mod}(n, p)$ ,  $q := p + r$  and  $w$  the suffix of  $v$  of length  $q$ . Then for all  $j$  in  $[1, n - q]$   $v_j = 1$  if  $j = ip$  for some  $i$ , and  $v_j = 0$  otherwise; and the following conditions hold:

1.  $r = 0$  or  $w_p = 1$ ,
2. if  $\pi(w) < p$  then  $\pi(w) + p > q + \text{gcd}(\pi(w), p)$ ,
3.  $w$  satisfies predicate  $\Xi$ .

*Case b:*  $p > \lfloor n/2 \rfloor$ . We have  $\forall j: 1 \leq j < p, v_j = 0$ . Let  $w$  be the suffix of  $v$  of length  
661  $n - p$ , then  $w$  satisfies predicate  $\Xi$ .

662 ► **Theorem 19.** *Let  $v$  a binary string of length  $n$ . The following statements are equivalent:*

- 663 1.  $v$  is the autocorrelation of a binary word
- 664 2.  $v$  is the autocorrelation of a word over an alphabet of size  $\geq 2$
- 665 3.  $v_0 = 1$  and  $v$  satisfies the **Forward and Backward Propagation Rules**
- 666 4.  $v$  satisfies the predicate  $\Xi$ .

667 Let  $v \in \{0, 1\}^n$ . We state the original definitions of FPR and BPR.

668 ► **Definition 20.**  $v$  satisfies the FPR iff for all pairs  $(p, q)$  satisfying  $0 \leq p < q < n$  and  
669  $v_p = v_q = 1$ , it follows that  $v_{p+i(q-p)} = 1$  for all  $i = 2, \dots, \lfloor (n-p)/(q-p) \rfloor$ .

670 ► **Definition 21.**  $v$  satisfies the BPR iff for all pairs  $(p, q)$  satisfying  $0 \leq p < q < 2p$ ,  
671  $v_p = v_q = 1$ , and  $v_{2p-q} = 0$ , it follows that  $v_{p-i(q-p)} = 0$  for all  $i = 2, \dots, \min(\lfloor p/(q-p) \rfloor$   
672  $\lfloor (n-p)/(q-p) \rfloor)$ .